**Class Diagram**

+--------------------+

| Bank |

+--------------------+

| - bankName: String |

| - branches: Array |

+--------------------+

| + addBranch() |

| + getBranches() |

+--------------------+

|

| has

|

+---------------------+

| Branch |

+---------------------+

| - branchName: String|

| - branchCode: String|

| - accounts: Array |

+---------------------+

| + addAccount() |

| + getAccounts() |

+---------------------+

|

| has

|

+-----------------------+ +------------------------+

| Account |<-------| SavingsAccount |

+-----------------------+ +------------------------+

| - accountNumber: Int | | - interestRate: Float |

| - accountHolder: String| +------------------------+

| - balance: Float | | + calculateInterest() |

+-----------------------+ +------------------------+

| + deposit() |

| + withdraw() |

| + checkBalance() |

+-----------------------+

|

|

|

+------------------------+

| CurrentAccount |

+------------------------+

| - overdraftLimit: Float|

+------------------------+

| + checkOverdraft() |

+------------------------+

**Explanation of the Classes:**

1. **Bank Class**:
   * Represents a **bank**.
   * Attributes:
     + bankName: Name of the bank.
     + branches: An array of Branch objects representing the branches of the bank.
   * Methods:
     + addBranch(branch): Adds a branch to the bank.
     + getBranches(): Returns all the branches of the bank.
2. **Branch Class**:
   * Represents a **branch** of the bank.
   * Attributes:
     + branchName: Name of the branch.
     + branchCode: Unique code identifying the branch.
     + accounts: An array to store accounts (both SavingsAccount and CurrentAccount).
   * Methods:
     + addAccount(account): Adds an account to the branch.
     + getAccounts(): Returns all the accounts in the branch.
3. **Account Class** (Base class):
   * Represents a general **bank account**.
   * Attributes:
     + accountNumber: The unique number for the account.
     + accountHolder: The name of the account holder.
     + balance: The balance of the account (defaults to 0).
   * Methods:
     + deposit(amount): Deposits an amount into the account.
     + withdraw(amount): Withdraws an amount from the account if sufficient balance is available.
     + checkBalance(): Displays the current balance of the account.
4. **SavingsAccount Class** (Inherits from Account):
   * Represents a **savings account**, which is a type of bank account that earns interest.
   * Attributes:
     + interestRate: The interest rate for the savings account.
   * Methods:
     + calculateInterest(): Calculates and prints the interest earned based on the current balance and interest rate.
5. **CurrentAccount Class** (Inherits from Account):
   * Represents a **current account**, which may have an overdraft limit.
   * Attributes:
     + overdraftLimit: The maximum limit up to which the account holder can withdraw beyond their balance (negative balance).
   * Methods:
     + checkOverdraft(): Checks if the account balance is negative and if it is within the overdraft limit.

**Relationships:**

* **Bank to Branch**: A bank contains multiple branches (1 to many relationship).
* **Branch to Account**: A branch contains multiple accounts (1 to many relationship).
* **Account to SavingsAccount/CurrentAccount**: SavingsAccount and CurrentAccount inherit from the base Account class, thus representing a "is-a" relationship.

**Flow of the Bank System:**

1. **Bank** adds multiple **branches**.
2. Each **branch** can have multiple **accounts** (both SavingsAccount and CurrentAccount).
3. **Account** is a base class with common attributes and methods for deposit, withdrawal, and balance check.
   * **SavingsAccount** extends **Account** and adds functionality for interest calculation.
   * **CurrentAccount** extends **Account** and adds functionality for overdraft checking.
4. A user can **deposit**, **withdraw**, or **check the balance** of their account.
5. A **SavingsAccount** user can calculate and earn interest.
6. A **CurrentAccount** user can check if they are eligible for overdraft services.

**Suggested Implementation for Students:**

1. **Account Class**: Implement the basic structure for an account. This will be the foundation for both savings and current accounts.
2. **SavingsAccount Class**: Extend the Account class, add an interestRate property, and a method to calculate interest.
3. **CurrentAccount Class**: Extend the Account class, add an overdraftLimit property, and implement overdraft checking.
4. **Branch Class**: Create a branch class that can hold multiple accounts, with methods to add and retrieve accounts.
5. **Bank Class**: Implement the bank class that can manage multiple branches, adding and retrieving branches.

By having students implement these classes, they'll practice object-oriented concepts like inheritance, class relationships, and encapsulation, and will understand how these classes work together to form a real-world banking system.

Would you like to provide more examples or details for the students?